Busca em Vetores

Busca em Vetores

- A busca nada mais é do que o ato de procurar um elemento em um conjunto de dados.
- A operação de busca visa responder se determinado valor está ou não presente em um conjunto de elementos (por exemplo, um array).
- No caso do item que se busca presente no conjunto de elementos, seus dados são retornados para o usuário. Toda busca é feita utilizando como base uma chave específica.
- A chave de busca é o “campo” do item utilizado para comparação. É por meio dele que sabemos se dado elemento é o que buscamos.

Busca sequencial ou linear

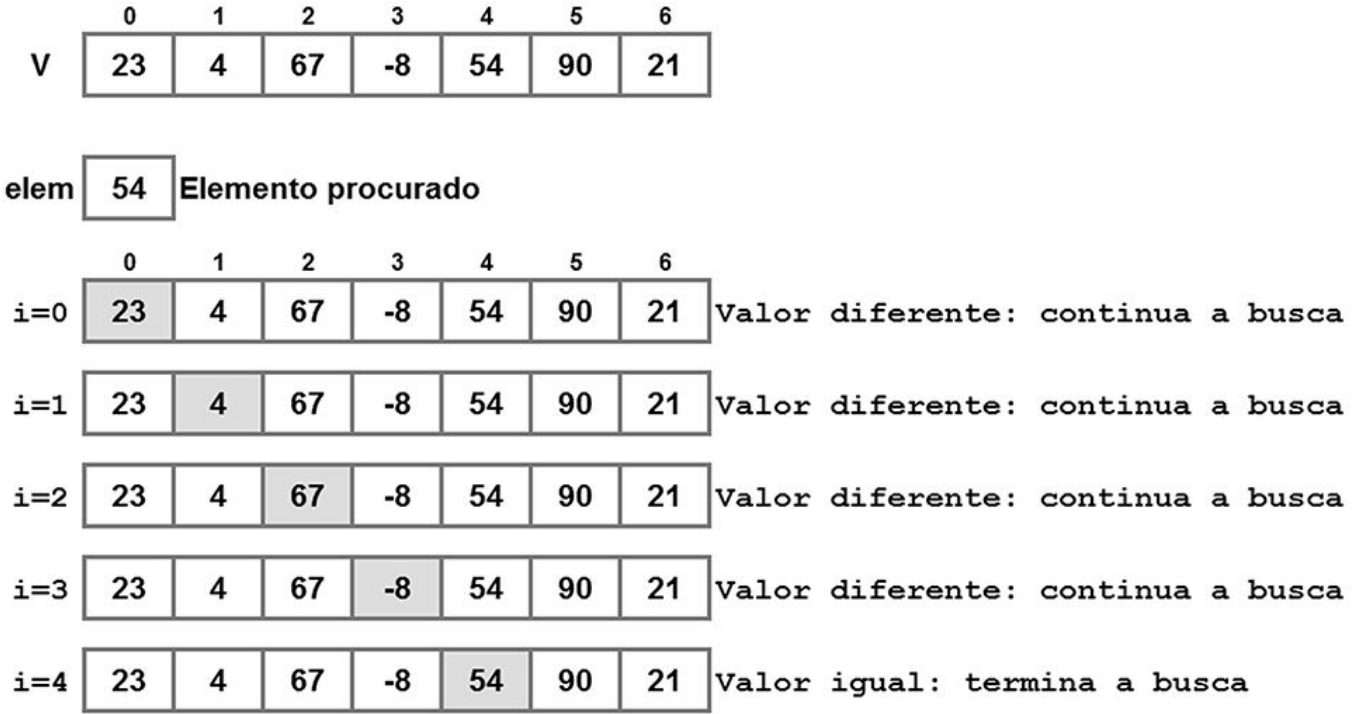
- A busca sequencial assume que os dados não estão ordenados, por isso a necessidade de percorrer o array do seu início até o seu fim.

Busca sequencial ou linear

```
01  int buscaLinear(int *V, int N, int elem){
02      int i;
03      for(i = 0; i<N; i++){
04          if(elem == V[i])
05              return i;//elemento encontrado
06      }
07      return -1;//elemento não encontrado
08  }
```

Busca sequencial ou linear

- A busca sequencial assume que os dados não estão ordenados, por isso a necessidade de percorrer o array do seu início até o seu fim.



Busca sequencial ou linear

- A busca sequencial assume que os dados não estão ordenados, por isso a necessidade de percorrer o array do seu início até o seu fim.
    - Quanto tempo demora para executar esse algoritmo de busca?

Considerando um array com **N** elementos:

- $O(1)$, melhor caso: o elemento é o primeiro do array.
- $O(N)$, pior caso: o elemento é o último do array ou não existe.
- $O(N/2)$, caso médio.

Busca sequencial ou linear Ordenada

- A busca sequencial ordenada assume que os dados estão ordenados. Assim, se o elemento procurado for menor do que o valor em determinada posição do array, temos a certeza de que ele não estará no restante do array. Isso evita a necessidade de percorrer o array do seu início até o seu fim.

Busca sequencial ou linear ordenada

```
01  int buscaOrdenada(int *V, int N, int elem){
02      int i;
03      for(i = 0; i<N; i++){
04          if(elem == V[i])
05              return i;//elemento encontrado
06          else
07              if(elem < V[i])
08                  return -1;//parar a busca
09      }
10      return -1;//elemento não encontrado
11  }
```

Busca sequencial ou linear Ordenada

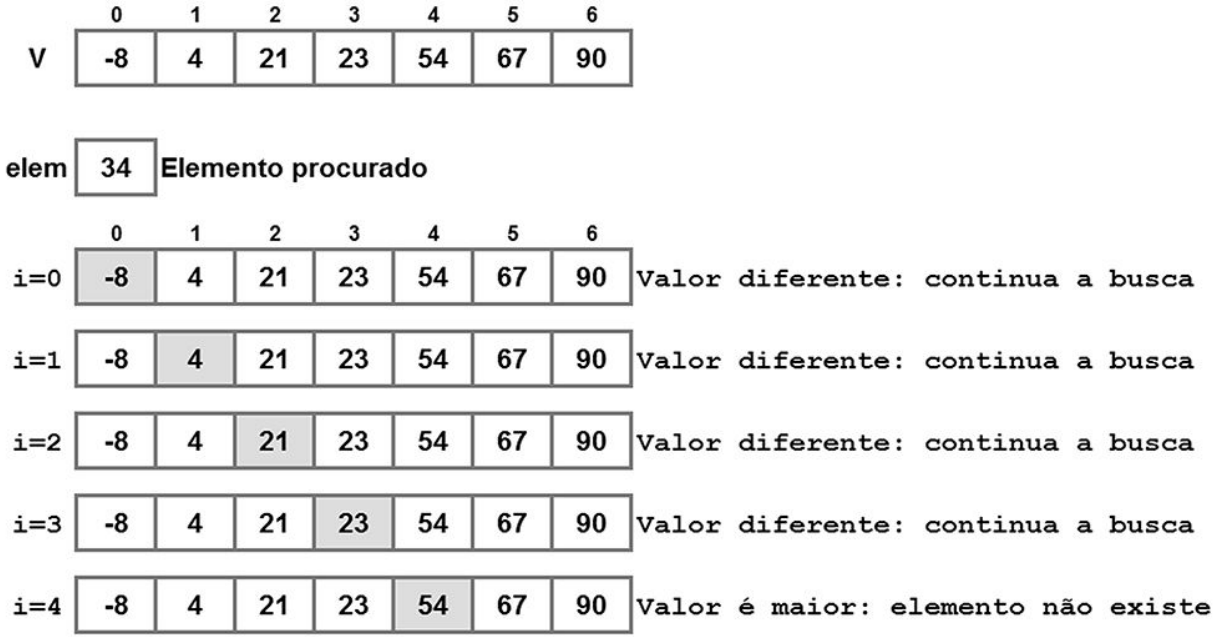
- A busca sequencial ordenada assume que os dados estão ordenados. Assim, se o elemento procurado for menor do que o valor em determinada posição do array, temos a certeza de que ele não estará no restante do array. Isso evita a necessidade de percorrer o array do seu início até o seu fim.

Busca sequencial ou linear ordenada

```
01  int buscaOrdenada(int *V, int N, int elem){
02      int i;
03      for(i = 0; i<N; i++){
04          if(elem == V[i])
05              return i;//elemento encontrado
06          else
07              if(elem < V[i])
08                  return -1;//parar a busca
09      }
10      return -1;//elemento não encontrado
11  }
```

Busca sequencial ou linear Ordenada

- A busca sequencial ordenada assume que os dados estão ordenados. Assim, se o elemento procurado for menor do que o valor em determinada posição do array, temos a certeza de que ele não estará no restante do array. Isso evita a necessidade de percorrer o array do seu início até o seu fim.



Busca Binária

- A busca binária é uma estratégia baseada na ideia de *dividir para conquistar*. A cada passo, esse algoritmo analisa o valor do meio do array. Caso o valor seja igual ao elemento procurado, a busca termina. Do contrário, a busca continua na metade do array que condiz com o valor procurado.

Busca binária

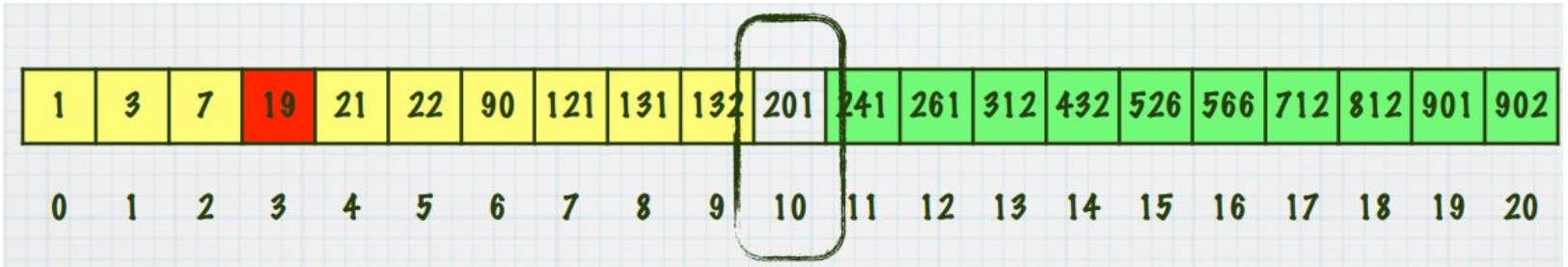
```
01  int buscaBinaria(int *V, int N, int elem){
02      int i, inicio, meio, final;
03      inicio = 0;
04      final = N-1;
05      while(inicio <= final){
06          meio = (inicio + final)/2;
07          if(elem < V[meio])
08              final = meio-1;//busca na metade da esquerda
09          else
10              if(elem > V[meio])
11                  inicio = meio+1;//busca na metade direita
12              else
13                  return meio;
14      }
15      return -1;//elemento não encontrado
16  }
```

Busca Binária

- A busca binária é uma estratégia baseada na ideia de *dividir para conquistar*. A cada passo, esse algoritmo analisa o valor do meio do array. Caso o valor seja igual ao elemento procurado, a busca termina. Do contrário, a busca continua na metade do array que condiz com o valor procurado.

O problema da busca

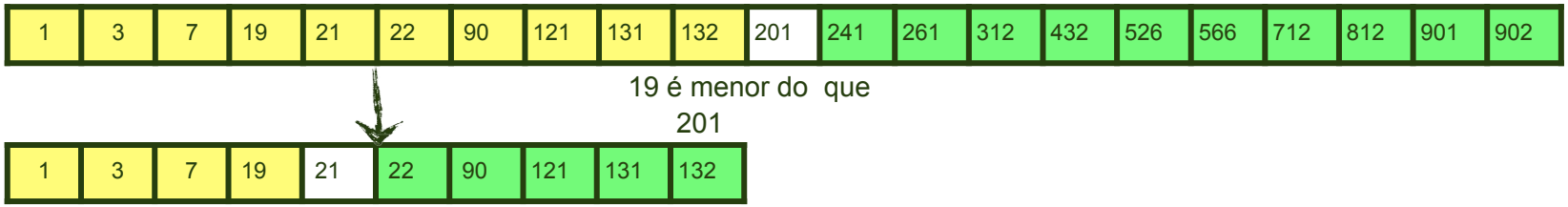
- Divisão e conquista
  - Array ordenado
  - Verifica se elemento está no meio do array Como o
  - array está ordenado:
    - Se for menor que o do meio, está a esquerda
    - Se for maior está a direita
- Busca do elemento 19



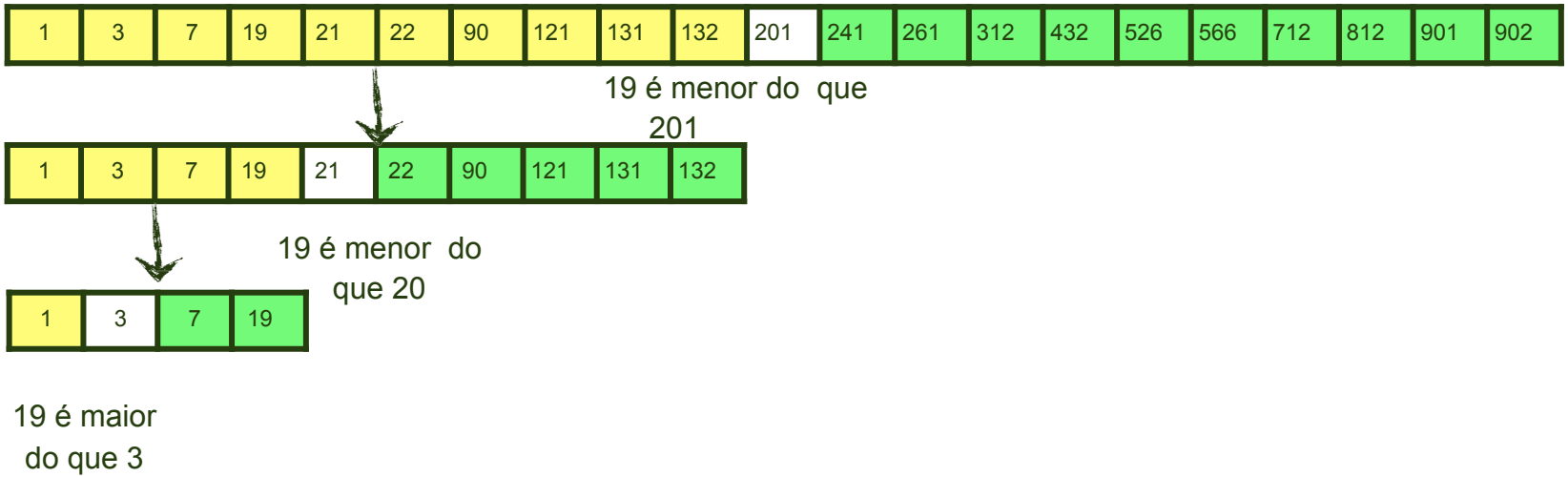
O problema da busca

19 é menor do que 201

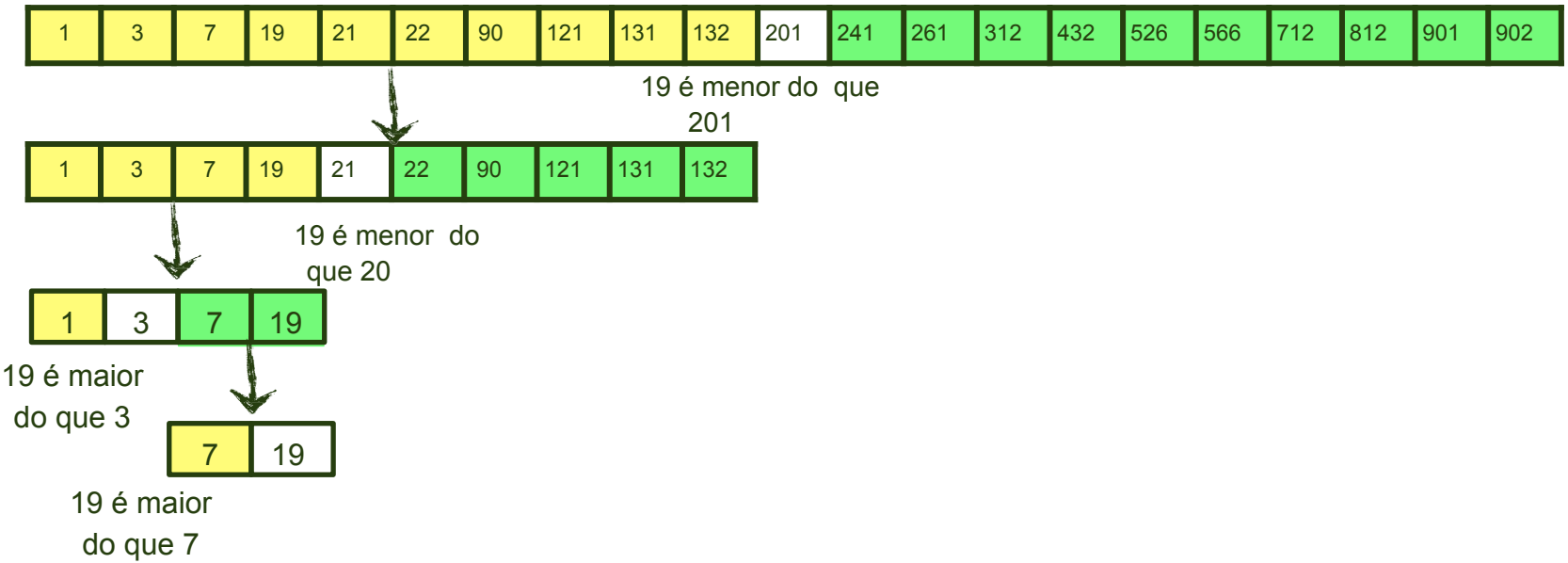
O problema da busca



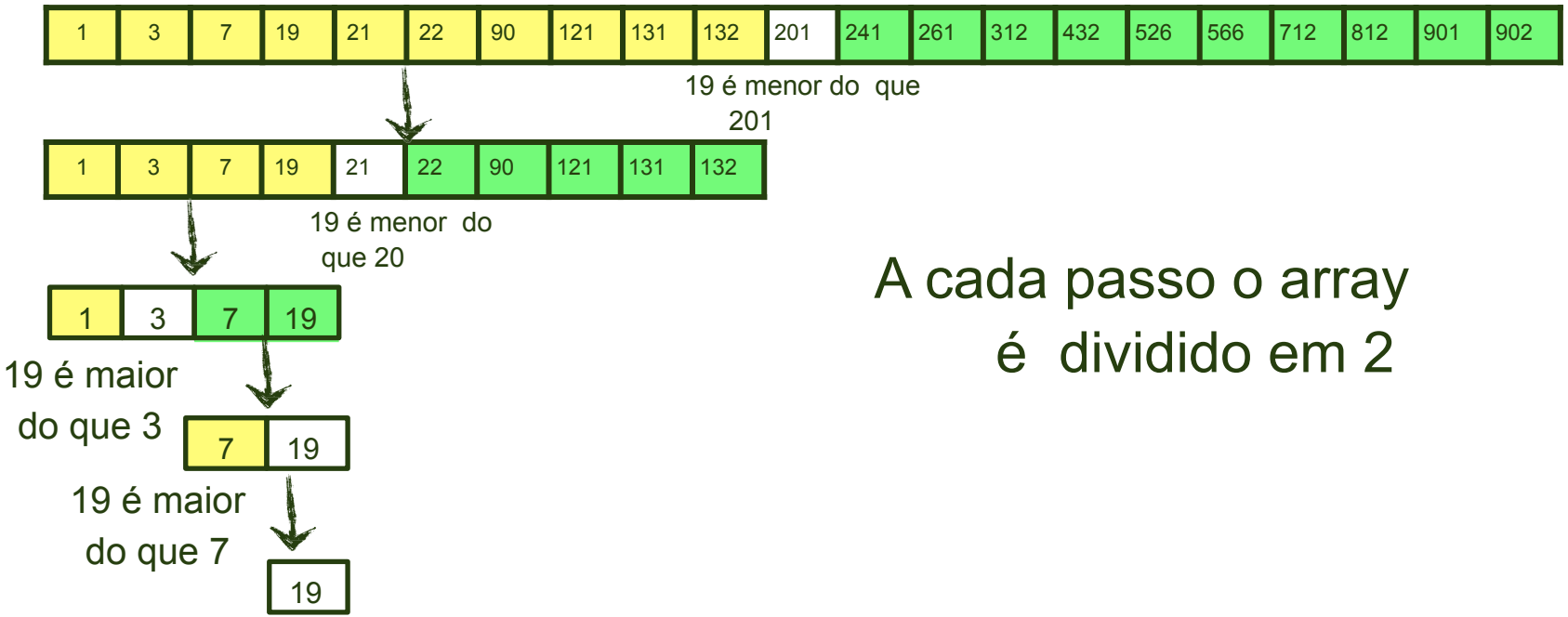
O problema da busca



O problema da busca



O problema da busca



A cada passo o array é dividido em 2

Busca Binária

- A busca binária é uma estratégia de busca muito mais eficiente do que a busca sequencial ordenada.

Iteração 1:

tamanho do vetor = n

Iteração 2:

tamanho do vetor = n/2

Iteração 3:

tamanho do vetor = (n/2)/2 = $n/2^2$

…

Iteração k:

tamanho do vetor = $n/2^k$

Busca Binária

- A busca binária é uma estratégia de busca muito mais eficiente do que a busca sequencial ordenada.

Iteração 1:

tamanho do vetor = n

Iteração 2:

tamanho do vetor = n/2

Iteração 3:

tamanho do vetor = (n/2)/2 = $n/2^2$

…

Iteração k:

tamanho do vetor = $n/2^k$

Também sabemos que após k iterações, o tamanho do vetor se torna 1. Portanto o tamanho do vetor se torna:

$n/2^k = 1$

$\Rightarrow \quad n=2^k$

aplicando log dos dois lados…

$\Rightarrow \log_2 n = \log_2 2^k$

$\Rightarrow \log_2 n = k * \log_2 2$

Portanto, $k=\log_2 n$

Busca Binária

- A busca binária é uma estratégia de busca muito mais eficiente do que a busca sequencial ordenada.

Considerando um array com **N** elementos, o tempo de execução da busca binária é:

- $O(1)$, melhor caso: o elemento procurado está no meio do array.
- $O(log_2 N)$, pior caso: o elemento não existe.
- $O(log_2 N)$, caso médio.

Atividade 1 - Busca Binária

Se preciso de t segundos para fazer uma busca binária em um vetor com n elementos, de quando tempo preciso para fazer uma busca em n^2 elementos?

Atividade 2 - Busca Binária

A função a seguir implementa a versão recursiva da busca binária. Complete os valores das lacunas e descreva a relação de recorrência do código.

```
int buscaR(int v[], int n, int procurado, int esq, int dir){

int meio = (esq+dir)/2;

if(esq<=dir){

        if(v[meio] == procurado)

                return 1;

        else if(v[meio] > procurado)

                return buscaR(v, n, procurado, _________, ________);

        else if(v[meio] < procurado)

                return buscaR(v, n, procurado, _________, ________);

        }

return 0;
```

Atividade 3 - Busca Binária

Elabore o jogo “Número Mágico” que “adivinha” o número que o usuário pensa. O usuário deve pensar em um número entre 0 e 10.000. O jogo deve utilizar a busca binária para interagir com o jogador e descobrir o número. Analise a complexidade assintótica (big-O), do código em função da quantidade de interações com o usuário.

Referências

- Estrutura de dados descomplicada em C (André Backes) - Cap. 3 - 3.6
- Projeto de Algoritmos (Ziviani) - Cap 5 - 5.1, 5.2